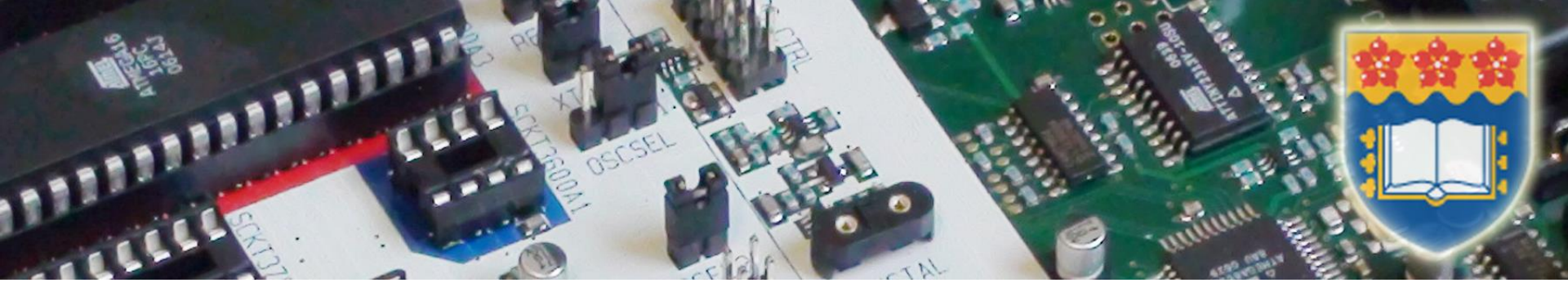


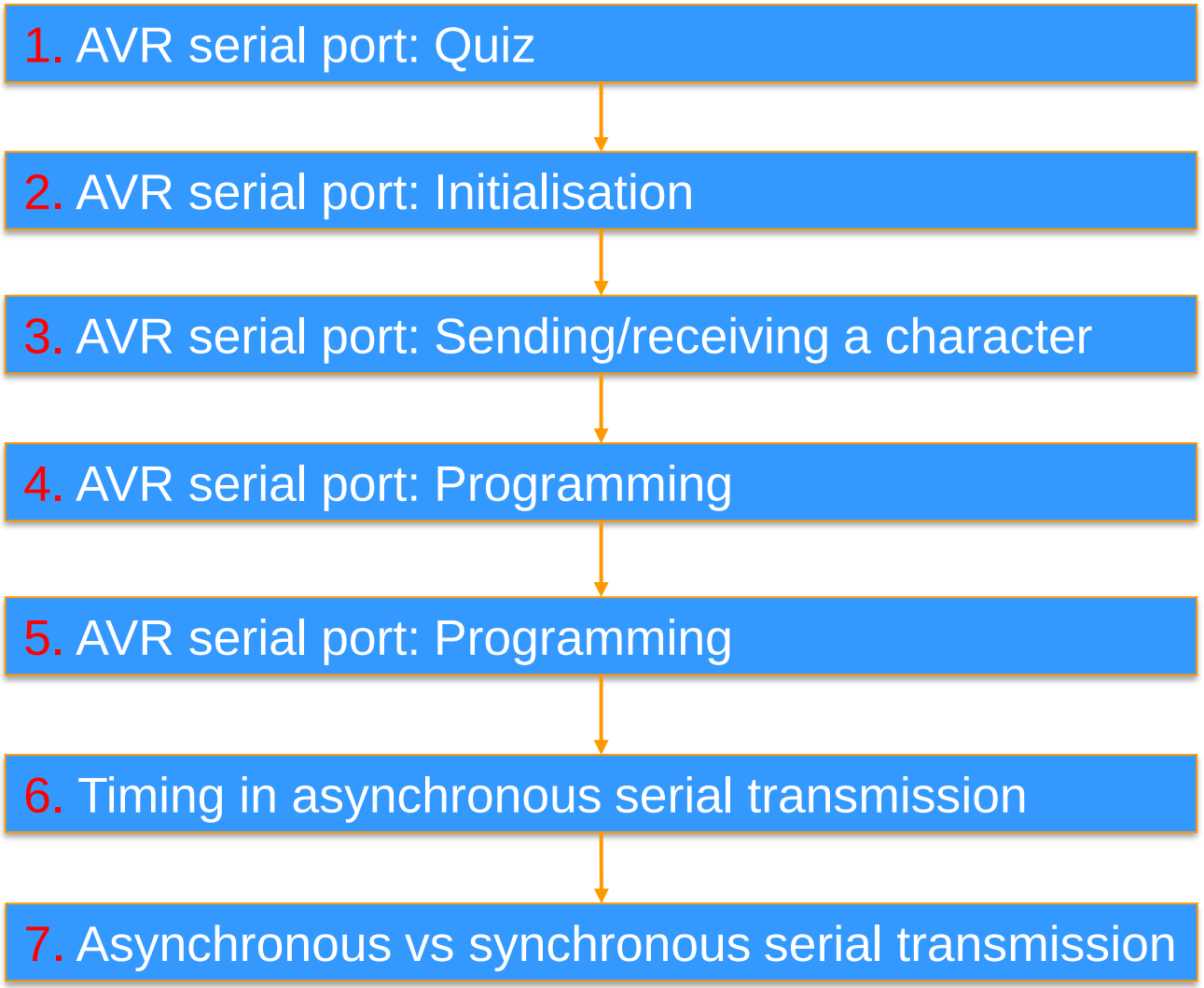
ECTE333 – Tutorial 2
Serial Communication

School of Electrical, Computer and Telecommunications Engineering
University of Wollongong
Australia

ECTE333's schedule

Lecture (2h)	Tutorial (1h)	Lab (2h)
L1:Introduction to AVR Microcontrollers		
L2: C Programming, Digital IO	Tutorial 1	Lab 1
L3: Serial Communication		
	Tutorial 2	Lab 2
L4: Interrupts, Timers		
	Tutorial 3	Lab 3
L5: Pulse Width Modulators		
	Tutorial 4	Lab 4
L6: Analogue-to-Digital Converters		
	Tutorial 5	Lab 5
L7: Microcontroller Applications		
		Lab 6

Tutorial 2's overview



Tutorial 2's overview

For this tutorial, we use

Section 5 of Exam Appendix:

[ECTE333-Exam-Appendix.pdf]

(available on e-learning)

5. Serial Communication

Registers UBRRH & UBRRL

URSEL→	0	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

$$baud\ rate = \frac{system\ clock\ frequency\ (Hz)}{16 \times (UBRR + 1)}; \qquad UBRR = \frac{system\ clock\ frequency\ (Hz)}{16 \times (baud\ rate)} - 1$$

Register UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

RXC: 1 when receive buffer has unread data (RX complete)
TXC: 1 when no new data in transmit buffer (TX complete)
UDRE: 1 when USART data register is empty
FE: 1 when there is a frame error
DOR: 1 when there is a data overrun error
PE: 1 when there is a parity error
U2X: 1 to double the transmission speed
MPCM: 1 to enable multi-processor com mode

Register UCSRB

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

RXCIE: 1 to enable RX Complete Interrupt, valid only if Global Interrupt Flag = 1 and RXC = 1
TXCIE: 1 to enable TX Complete Interrupt, valid only if Global Interrupt Flag = 1 and TXC =1
UDRIE: 1 to enable USART Data Register Empty Interrupt
RXEN: 1 to enable USART receiver: Pin D.0 = RXD pin
TXEN: 1 to enable USART transmitter: Pin D.1 = TXD pin
UCSZ2: bit UCSZ2 for selecting character size
RXB8: Rx extra data bit for 9-bit character size
TXB8: Tx extra data bit for 9-bit character size

Register UCSRC

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

URSEL: Must be set to 1 to write to UCSRC. Note: UCSRC and UBRRH share the same location
UMSEL: To select USART modes: 0 asynchronous, 1 synchronous
UPM1, UPM0: To select parity mode: 00 no parity, 10 even party, 11 odd parity
USBS: To select stop bit modes: 0→1 stop bit, 1→2 stop bit
UCSZ1, UCSZ0: Used with UCSZ2 to select character size
UCPOL: Clock polarity, used with synchronous

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Q1 – Serial communication in ATmega16: Quiz

a) On what IO port do TXD and RXD pins reside?

- Port D: TXD = D.1 and RXD = D.0.

b) What registers are used to set the baud rate?

- UBRRH and UBRRL.

$$UBRR = \frac{system\ clock\ frequency\ (Hz)}{16 \times (baud\ rate)} - 1$$

c) How to select asynchronous transmission?

- Make flag UMSEL = 0 (register UCSRC).

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

UCSRC

d) How to specify the number of stop bits?

- Flag USBS of register UCSRC: 0 → 1 stop bit,
  1 → 2 stop bits.

Q1 – Serial communication in ATmega16: Quiz

e) How to select parity bit options?

- Bits UPM1 and UPM0 of register UCSRC: 00 → none, 10 → even, 11 → odd.

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

UCSRC

f) How to double the transmission speed?

- Set bit U2X = 1 (register UCSRA).
- Then, baud rate = clock/[8 ×(UBRR+1)].

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

UCSRA

g) How to specify the character size 5, 6, 8, or 9?

- Use three flags UCSZ2, UCSZ1 and UCSZ0.

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

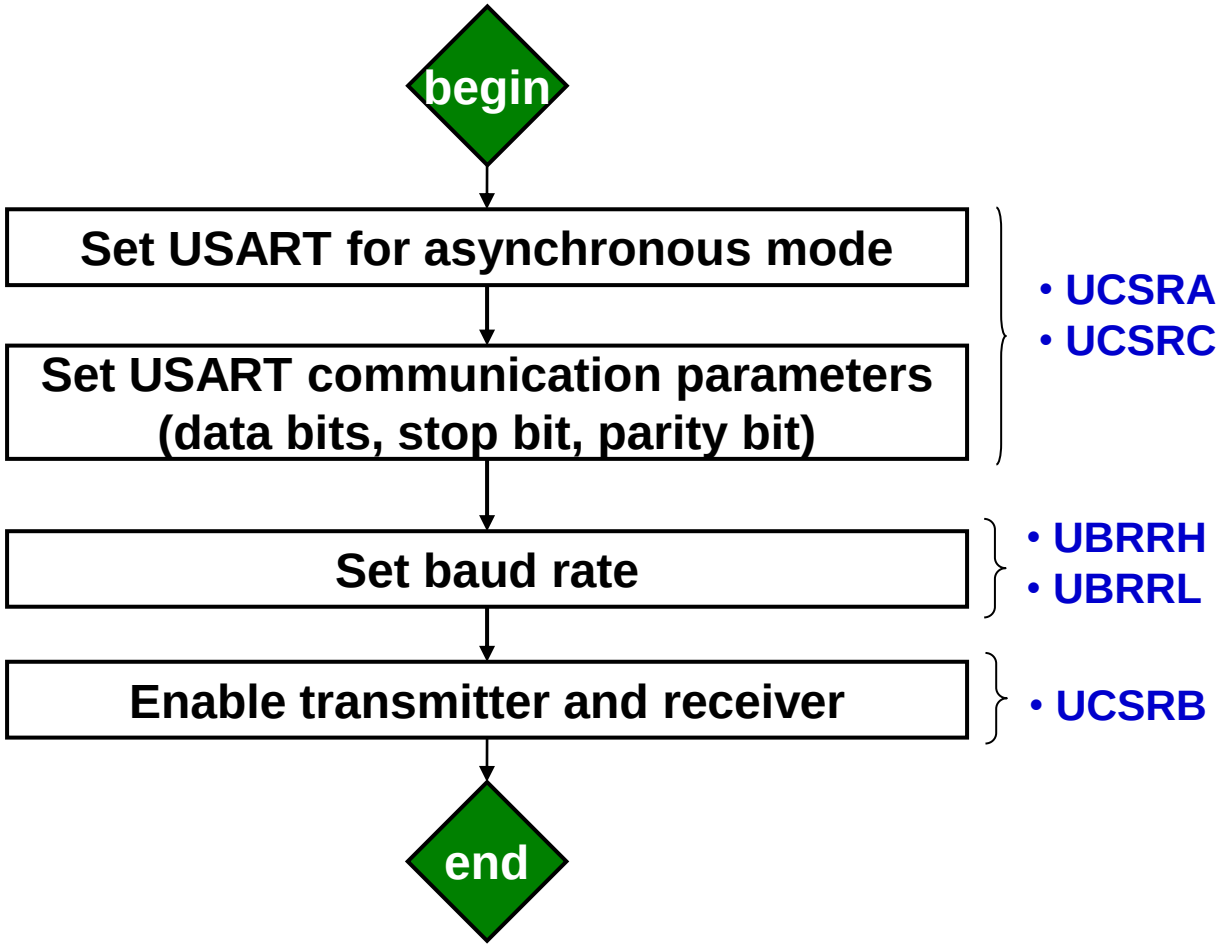
UCSRB

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

UCSRC

Q2 – Serial communication in ATmega16: Initialisation

a) Describe the steps and the relevant registers to initialise the serial port on the ATmega16 for asynchronous serial transmission.



Q2 – Serial communication in ATmega16: Initialisation

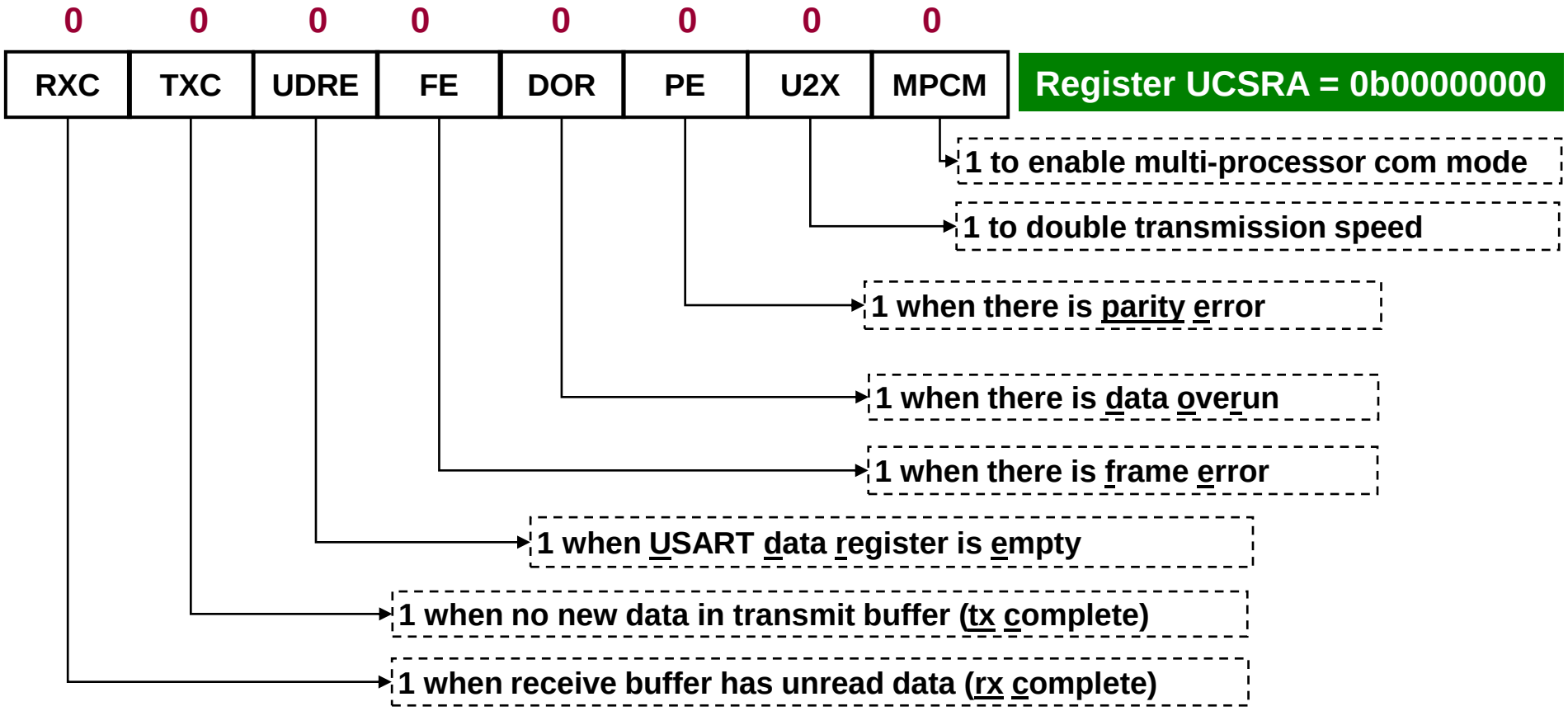
b-i) Write C code for asynchronous serial transmission at baud rate 2400bps, 8 data bits, 1 start bit, 1 stop bit, no parity, system clock 1MHz.

- **Baud rate**
  - UBRR = 1000000/(16 × 2400) – 1 = 25_d = 0019_H.
  - Therefore, UBRRH = 00_H and UBRRL = 19_H.
- **Control and status registers** (next 3 slides)
  - UCSRA = 0b0000 0000
  - UCSRB = 0b0001 1000
  - UCSRC = 0b1000 0110
- **C code**

```
UCRSA = 0b00000000;
UCSRB = 0b00011000;
UCSRC = 0b10000110;
UBRRH = 0x00; UBRRL = 0x19;
```

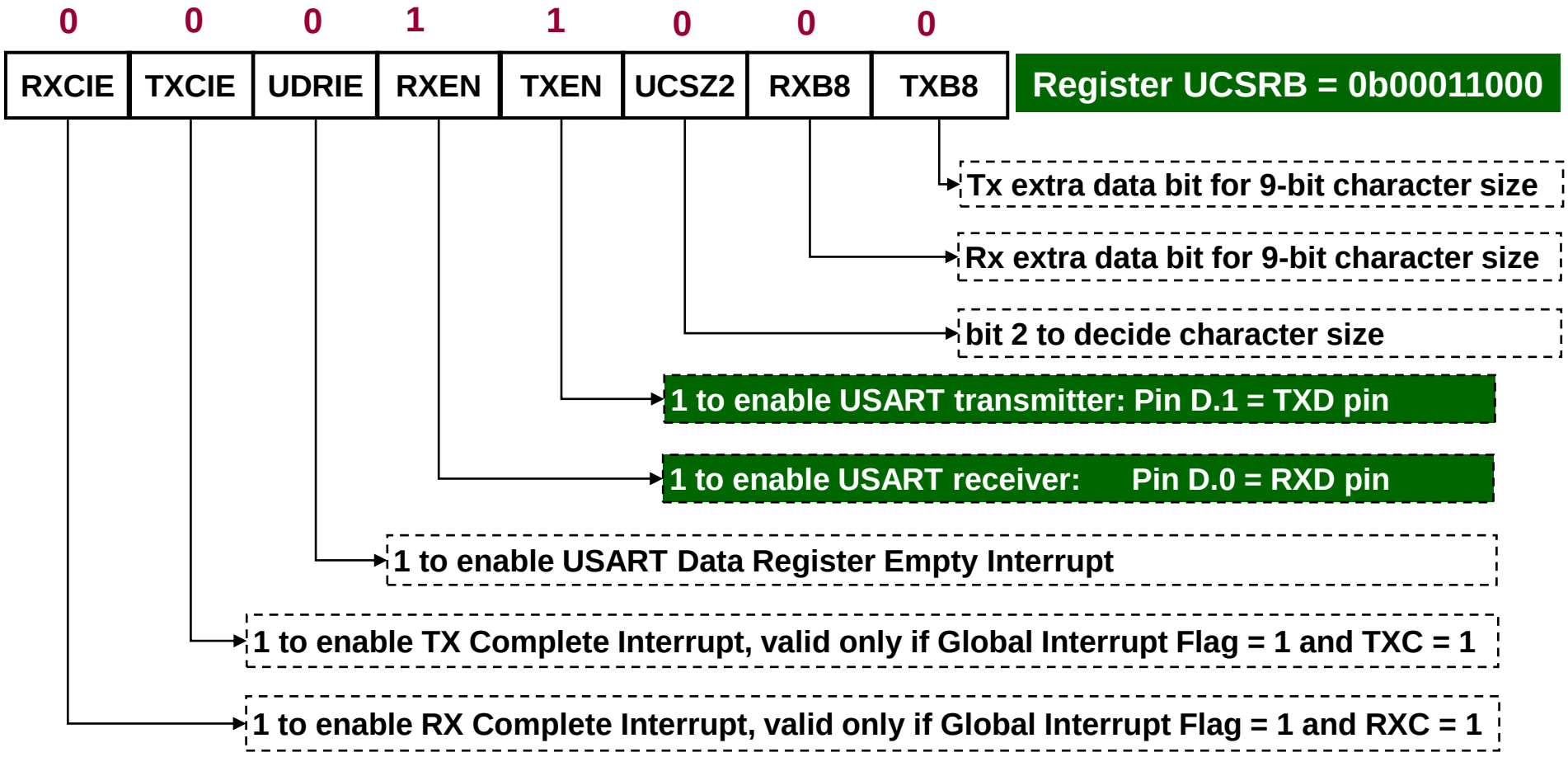
Q2 – Serial communication in ATmega16: Initialisation

b-i) Write C code for asynchronous serial transmission at baud rate 2400bps, 8 data bits, 1 start bit, 1 stop bit, no parity, system clock 1MHz.



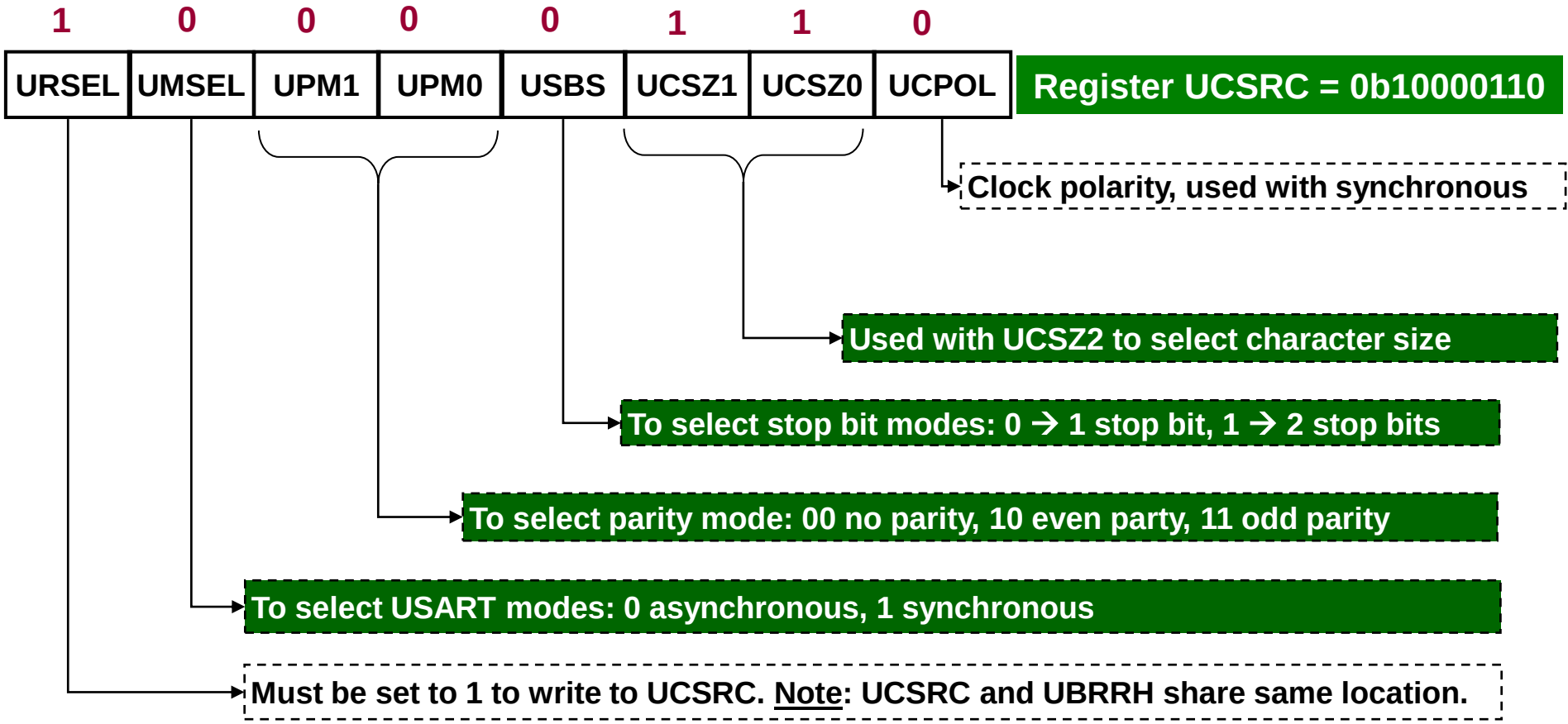
Q2 – Serial communication in ATmega16: Initialisation

b-i) Write C code for asynchronous serial transmission at baud rate 2400bps, 8 data bits, 1 start bit, 1 stop bit, no parity, system clock 1MHz.



Q2 – Serial communication in ATmega16: Initialisation

b-i) Write C code for asynchronous serial transmission at baud rate 2400bps, 8 data bits, 1 start bit, 1 stop bit, no parity, system clock 1MHz.



Q2 – Serial communication in ATmega16: Initialisation

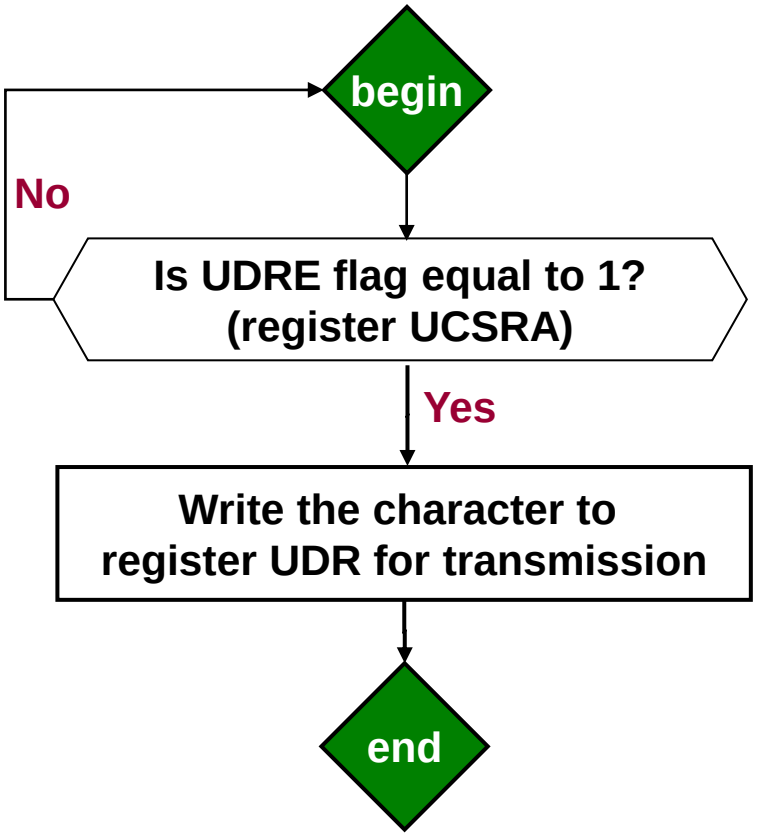
b-ii) Write C code for asynchronous serial transmission at baud rate 4800bps, 7 data bits, 1 start bit, 2 stop bits, even parity, system clock 4 MHz.

- **C code**

```
UCRSA = 0b00000000;
UCSRB = 0b00011000;
UCSRC = 0b10101100;
UBRRH = 0x00; UBRRL = 0x33;
```

Q3 – Serial communication in ATmega16: Sending a character

a) Explain the steps and the registers involved in sending a character through the serial port. Assume that the polling approach is used.



	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
UCSRA								
mask	0	0	1	0	0	0	0	0
bit-wise AND	0	0	UDRE	0	0	0	0	0

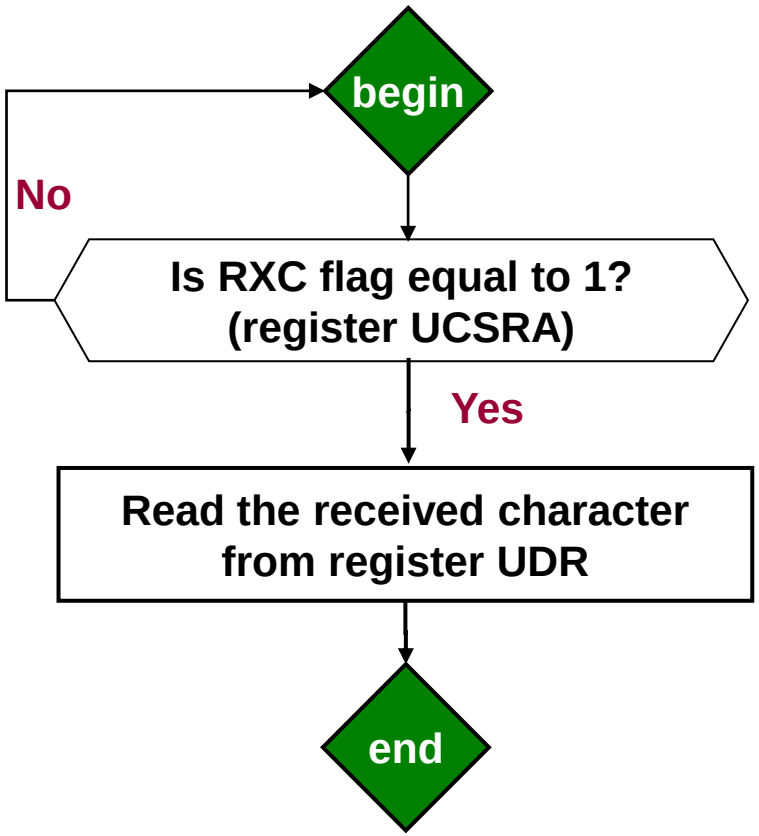
zero if bit at position UDRE = 0

```
void serial_send(unsigned char data){
   // Wait until UDRE flag = 1
   while ((UCSRA & 0b00010000) == 0x00){;}

   // Write char to UDR for transmission
   UDR = data;
}
```

Q3 – Serial communication in ATmega16: Receiving a character

b) Explain the steps and the registers involved in receiving a character through the serial port. Assume that the polling approach is used.



UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
mask	1	0	0	0	0	0	0	0
bit-wise AND	RXC	0	0	0	0	0	0	0

zero if bit at position RXC = 0

```
unsigned char serial_receive(void){
   // Wait until RXC flag = 1
   while ((UCSRA & 0b10000000) == 0x00){;}

   // Read the received char from UDR
   return (UDR);
}
```

Q4 – Serial communication in ATmega16: Programming

Write a C program to continually read a character from the serial port of ATmega16 and send the character to PORTA. Use parameters in Q2b-i.

```
#include <avr/io.h>
void serial_init(void){
    UCSRA = 0b00000000; // disable double speed, disable multi-proc
    UCSRB = 0b00011000; // enable Tx and Rx, disable interrupts
    UCSRC = 0b10000110; // asyn mode, no parity, 1 stop bit, 8 data bits
    UBRRH = 0x00; UBRRL = 0x19; // Baud rate 2400bps, assuming 1MHz clock
}
unsigned char serial_receive(void){
    while ((UCSRA & (1<<RXC)) == 0x00){;} // wait until RXC flag = 1
    return (UDR);                 // read the received character from UDR
}

int main(void) {
    unsigned char i;
    DDRA = 0xFF;    // set PORTA for output
    serial_init(); // initialise USART
    while (1) {
        i = serial_receive(); // receive from serial port
        PORTA = i;            // send it to port A
    }
    return 0;
}
```

Q5 – Serial communication in ATmega16: Programming

Write a C program to control a pan-tilt camera through the serial port …

```
#include <avr/io.h>
void delay(void){
   for (int i = 0; i < 1000; i++)
      for (int j = 0; j < 100; j++)
        asm volatile("nop"); // create a relay by repeating NOP instructions
}

void serial_init(void){
   UCSRA = 0b00000010; // double speed, disable multi-proc
   UCSRB = 0b00011000; // Enable Tx and Rx, disable interrupts
   UCSRC = 0b10000110; // Asyn mode, no parity, 1 stop bit, 8 data bits
   UBRRH = 0x00; UBRRL = 12; // Baud rate 9600bps, 1MHz clock, double-speed
}

void serial_send(unsigned char data){
   while ((UCSRA & (1<<UDRE)) == 0x00){;} // wait until UDRE flag = 1
   UDR = data; // Write character to UDR for transmission
}
int main(void) {
   unsigned char i, j;
   unsigned char move_chars[4] = {'4', '6', '8', '2'};
   serial_init(); // initialise USART
   while (1) {
      for (j = 0; j<4; j++)
         for (i = 0; i < 5; i++){        // rotate 5 times
            serial_send(move_chars[j]); // send '4', '6', '8', or '2'
            delay();
         }
    }
    return 0;
}
```

Q6 – Timing in asynchronous serial transmission

- Asynchronous serial transmission uses 8 data bits, 1 start bit, 1 stop bit and 1 parity bit.
- Incoming bits are sampled at the middle of each clock period of the receiver.
- Determine the relationship between the clock frequency of receiver & sender to prevent framing errors.

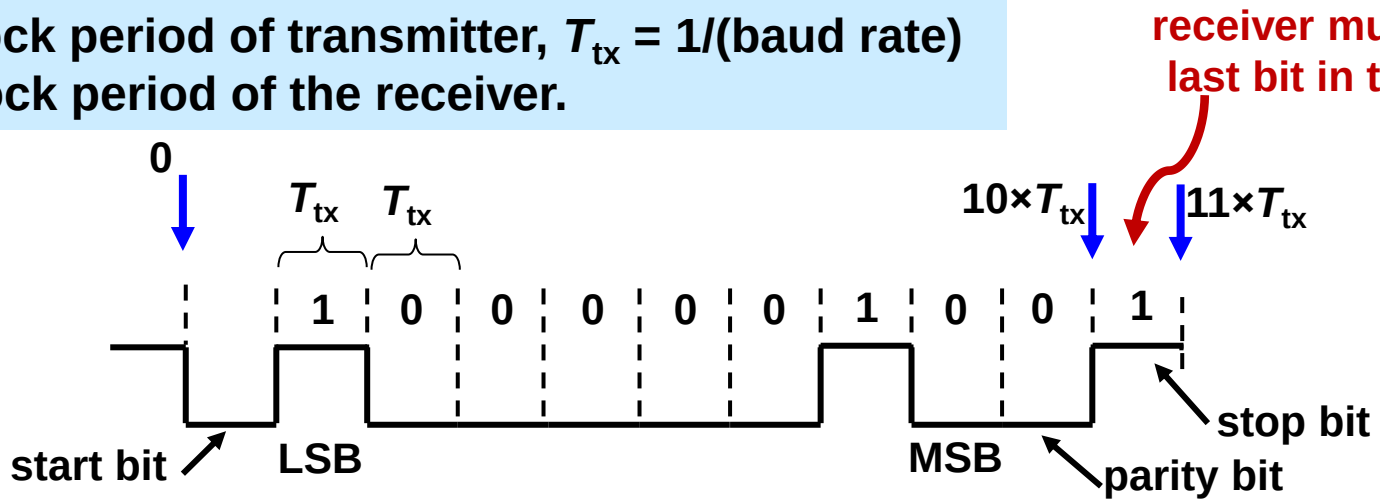
- Main idea: In asynchronous serial transmission, the clock rates of the sender and receiver do not have to be the same.

- Each character requires 11 bits to send: 8 data + 1 start + 1 stop + 1 parity.
- The receiver reads the 11th bit to find the end of transmission.
- If it cannot detect this bit, a framing error will occur.

- Let T_{tx} be the clock period of transmitter, T_{tx} = 1/(baud rate).
- Let T_{rx} be the clock period of the receiver.

Q6 – Timing in asynchronous serial transmission

T_{tx}: clock period of transmitter, T_{tx} = 1/(baud rate)
T_{rx}: clock period of the receiver.



- The transmitter sends the last bit (bit 11th) during: $[10 \times T_{tx}, 11 \times T_{tx}]$.
- The receiver reads the last bit (bit 11th) at time: $10.5 \times T_{rx}$.
- For the last bit to be read correctly, we require:

$$10 \times T_{tx} \leq 10.5 \times T_{rx} \leq 11 \times T_{tx}$$

$$\frac{10}{F_{tx}} \leq \frac{10.5}{F_{rx}} \leq \frac{11}{F_{tx}}$$

$$0.95 \times F_{tx} \leq F_{rx} \leq 1.05 \times F_{tx}]$$

Q7 – Asynchronous versus synchronous serial transmission

a) Differentiate synchronous versus asynchronous serial transmission.

- **Synchronous**
  - The clocks of the sender and receiver are synchronised.
  - A block of characters, enclosed by synchronising bytes, is sent at a time.
  - Faster transfer and less overhead.
  - Examples: SPI, BISYNC.
- **Asynchronous**
  - The clocks of the sender and receiver are not synchronised.
  - One character is sent at a time, enclosed by start bit, stop bits, & parity bit.
  - Example: asynchronous mode of serial port ATmega16.
  - For long messages, asynchronous mode is slow because start/stop bit must be sent for each character.
  - For short, interactive messages, asynchronous mode is suitable.

Q7 – Asynchronous versus synchronous serial transmission

b) Suppose a file of 10,000 bytes is to be sent over a line at 19600bps.

i. Calculate the overhead in bits and seconds for asynchronous mode. Assume 1 start bit, 1 stop bit, 8 data bits, and no parity bit.

ii. Calculate the overhead in bits and seconds for synchronous mode. Data are sent in frames: 1000 characters/frame, and an overhead of 48 bits/frame.

- **Asynchronous:** - Each character has 2 bits of overhead (1 start bit, 1 stop bit).
  - This means an extra of 20,000 bits are sent.
  - This leads to an overhead of 20,000/19,600 = 1.02 seconds.

- **Synchronous:**
  - 10 frames are required to transmit this file.
  - The control bits are the extra bits: 48 bits/frame × 10 frames = 480 bits.
  - The overhead time: 0.0245 seconds.

Extra practice for Lecture 3/Lab 3/Tutorial 2

- Write and test a C program for STK500 board that connects to the PC via serial port at baud rate 2400bps, 8 data bits, 1 start bit, 1 stop bit, no parity bit.

- The program should repeatedly
  - let the user enter two integer values from Hyper Terminal (using scanf).
  - turn ON or OFF the corresponding LED.

- The first integer is LED number, and the second integer is LED status.

  For example,
  - if the user enters 2 1, then LED2 will be turned ON.
  - If the user enters 2 0, then LED2 will be turned OFF.